Package 'FLAMES'

October 31, 2025

Title FLAMES: Full Length Analysis of Mutations and Splicing in long read RNA-seq data

Version 2.5.1

Date 2025-07-22

Description Semi-supervised isoform detection and annotation from both bulk and single-cell long read RNA-seq data. Flames provides automated pipelines for analysing isoforms, as well as intermediate functions for manual execution.

biocViews RNASeq, SingleCell, Transcriptomics, DataImport, DifferentialSplicing, AlternativeSplicing, GeneExpression, LongRead

BugReports https://github.com/mritchielab/FLAMES/issues

License GPL (>= 3)

Encoding UTF-8

Imports abind, basilisk, bambu, BiocParallel, Biostrings, BiocGenerics, crew, circlize, ComplexHeatmap, cowplot, cli, dplyr, GenomicRanges, GenomicFeatures, GenomicAlignments, Seqinfo, ggplot2, ggbio, grid, gridExtra, igraph, jsonlite, magrittr, magick, Matrix, MatrixGenerics, readr, reticulate, Rsamtools, rtracklayer, RColorBrewer, R.utils, S4Arrays, ShortRead, SingleCellExperiment, SummarizedExperiment, SpatialExperiment, scater, scatterpie, S4Vectors, scuttle, stats, scran, stringr, tidyr, utils, withr, methods, tibble, tidyselect, IRanges

Suggests BiocStyle, GEOquery, ggrastr, knitr, rmarkdown, uwot, testthat (>= 3.0.0), xml2

LinkingTo Rcpp, Rhtslib, testthat

SystemRequirements GNU make, C++17

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://mritchielab.github.io/FLAMES

Config/testthat/edition 3

Depends R (>= 4.2.0)

LazyLoad yes

StagedInstall no

git_url https://git.bioconductor.org/packages/FLAMES

git_branch devel

git_last_commit 7887658

git_last_commit_date 2025-10-31

Repository Bioconductor 3.23

Date/Publication 2025-10-31

Author Changqing Wang [aut, cre],
Luyi Tian [aut],
Oliver Voogd [aut],
Jakob Schuster [aut],
Shian Su [aut],
Yair D.J. Prawer [aut],
Yupei You [aut],
Matthew Ritchie [ctb]

Maintainer Changqing Wang <wang.ch@wehi.edu.au>

Contents

`addRowRanges` *Add rowRanges by rownames to* `SummarizedExperiment` *object Assumes rownames are transcript_ids Assumes transcript_id is present in the annotation file*

Description

Add rowRanges by rownames to `SummarizedExperiment` object Assumes rownames are transcript_ids Assumes transcript_id is present in the annotation file

Usage

```
addRowRanges(sce, annotation, outdir)
```

Value

a `SummarizedExperiment` object with rowRanges added

`add_gene_counts` *Add gene counts to a* `SingleCellExperiment` *object*

Description

Add gene counts to a `SingleCellExperiment` object as an `altExps` slot named gene.

Usage

```
add_gene_counts(sce, gene_count_file)
```

Arguments

`sce`	A `SingleCellExperiment` object.
`gene_count_file`	The file path to the gene count file. If missing, the function will try to find the gene count file in the output directory.

Value

A `SingleCellExperiment` object with gene counts added.

Examples

```
# Set up a mock SingleCellExperiment object
sce <- SingleCellExperiment::SingleCellExperiment(
  assays = list(counts = matrix(0, nrow = 10, ncol = 10))
)
colnames(sce) <- paste0("cell", 1:10)
# Set up a mock gene count file
gene_count_file <- tempfile()
gene_mtx <- matrix(1:10, nrow = 2, ncol = 5)
colnames(gene_mtx) <- paste0("cell", 1:5)
rownames(gene_mtx) <- c("gene1", "gene2")
write.csv(gene_mtx, gene_count_file)
# Add gene counts to the SingleCellExperiment object
sce <- add_gene_counts(sce, gene_count_file)
# verify the gene counts are added
SingleCellExperiment::altExps(sce)$gene
```

---

annotation_to_fasta *GTF/GFF to FASTA conversion*

Description

convert the transcript annotation to transcriptome assembly as FASTA file.

Usage

```
annotation_to_fasta(isoform_annotation, genome_fa, outfile, extract_fn)
```

Arguments

isoform_annotation	Path to the annotation file (GTF/GFF3)
genome_fa	The file path to genome fasta file.
outfile	The file path to the output FASTA file.
extract_fn	(optional) Function to extract a GRangesList object E.g. function(grl){GenomicFeatures::cdsBy(gr by="tx")}

Value

This does not return anything. A FASTA file will be created at the specified location.

Examples

```
fasta <- tempfile()
annotation_to_fasta(system.file("extdata", "rps24.gtf.gz", package = "FLAMES"), system.file("extdata", "rps24.fa
cat(readChar(fasta, 1e3))
```

blaze *BLAZE Assign reads to cell barcodes.*

Description

Uses BLAZE to generate barcode list and assign reads to cell barcodes.

Usage

```
blaze(expect_cells, fq_in, additional_args = NULL, ...)
```

Arguments

expect_cells	Integer, expected number of cells. Note: this could be just a rough estimate. E.g., the targeted number of cells.
fq_in	File path to the fastq file used as a query sequence file
additional_args	Additional command line style arguments to be passed to BLAZE. E.g. c("–10x-kit-version", "3v3")
...	Additional BLAZE configuration parameters. E.g., setting ‘’output-prefix’=’some_prefix’‘ is equivalent to specifying ‘–output-prefix some_prefix‘ in BLAZE; Similarly, ‘overwrite=TRUE‘ is equivalent to switch on the ‘–overwrite‘ option. Note that the specified parameters will override the parameters specified in the configuration file. All available options can be found at https://github.com/shimlab/BLAZE.

Value

A `data.frame` summarising the reads aligned. Other outputs are written to disk. The details of the output files can be found at https://github.com/shimlab/BLAZE.

Examples

```
outdir <- tempfile()
dir.create(outdir)
fastq <- system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES")
blaze(
  expect_cells = 10, fastq,
  "output-prefix" = file.path(outdir, ""),
  "output-fastq" = file.path(outdir, "output.fastq"),
  overwrite=TRUE
)
```

`BulkPipeline` *Pipeline for bulk long read RNA-seq data processing*

Description

Semi-supervised isofrom detection and annotation for long read data. This variant is meant for bulk samples. Specific parameters can be configured in the config file (see `create_config`), input files are specified via arguments.

Usage

```
BulkPipeline(
  config_file,
  outdir,
  fastq,
  annotation,
  genome_fa,
  genome_mmi,
  minimap2,
  samtools,
  controllers
)
```

Arguments

`config_file`	Path to the JSON configuration file. See `create_config` for creating one.
`outdir`	Path to the output directory. If it does not exist, it will be created.
`fastq`	Path to the FASTQ file or a directory containing FASTQ files. Each file will be processed as an individual sample.
`annotation`	The file path to the annotation file in GFF3 / GTF format.
`genome_fa`	The file path to the reference genome in FASTA format.
`genome_mmi`	(optional) The file path to minimap2's index reference genome.
`minimap2`	(optional) The path to the minimap2 binary. If not provided, FLAMES will use a copy from bioconda via `basilisk`.
`samtools`	(optional) The path to the samtools binary. If not provided, FLAMES will use a copy from bioconda via `basilisk`.
`controllers`	(optional, **experimental**) A `crew_class_controller` object for running certain steps

Details

By default FLAMES use minimap2 for read alignment. After the genome alignment step (`do_genome_align`), FLAMES summarizes the alignment for each read by grouping reads with similar splice junctions to get a raw isoform annotation (`do_isoform_id`). The raw isoform annotation is compared against the reference annotation to correct potential splice site and transcript start/end errors. Transcripts

that have similar splice junctions and transcript start/end to the reference transcript are merged with the reference. This process will also collapse isoforms that are likely to be truncated transcripts. If `isoform_id_bambu` is set to `TRUE`, `bambu::bambu` will be used to generate the updated annotations. Next is the read realignment step (`do_read_realign`), where the sequence of each transcript from the update annotation is extracted, and the reads are realigned to this updated `transcript_assembly.fa` by minimap2. The transcripts with only a few full-length aligned reads are discarded. The reads are assigned to transcripts based on both alignment score, fractions of reads aligned and transcript coverage. Reads that cannot be uniquely assigned to transcripts or have low transcript coverage are discarded. The UMI transcript count matrix is generated by collapsing the reads with the same UMI in a similar way to what is done for short-read scRNA-seq data, but allowing for an edit distance of up to 2 by default. Most of the parameters, such as the minimal distance to splice site and minimal percentage of transcript coverage can be modified by the JSON configuration file (`config_file`).

Value

A `FLAMES.Pipeline` object. The pipeline could be run using `run_FLAMES`, and / or resumed using `resume_FLAMES`.

See Also

`create_config` for creating a configuration file, `SingleCellPipeline` for single cell pipelines, `MultiSampleSCPipeline` for multi sample single cell pipelines.

Examples

```
outdir <- tempfile()
dir.create(outdir)
# simulate 3 samples via sampling
reads <- ShortRead::readFastq(
  system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES")
)
dir.create(file.path(outdir, "fastq"))
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample1.fq.gz"),
  mode = "w", full = FALSE
)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample2.fq.gz"),
  mode = "w", full = FALSE
)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads,
  file.path(outdir, "fastq/sample3.fq.gz"),
  mode = "w", full = FALSE
)
# prepare the reference genome
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
```

```
  destname = genome_fa, remove = FALSE
)
ppl <- BulkPipeline(
  fastq = c(
    "sample1" = file.path(outdir, "fastq", "sample1.fq.gz"),
    "sample2" = file.path(outdir, "fastq", "sample2.fq.gz"),
    "sample3" = file.path(outdir, "fastq", "sample3.fq.gz")
  ),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  genome_fa = genome_fa,
  config_file = create_config(outdir, type = "sc_3end", threads = 1, no_flank = TRUE),
  outdir = outdir
)
ppl <- run_FLAMES(ppl) # run the pipeline
experiment(ppl) # get the result as SummarizedExperiment
```

`bulk_long_pipeline` *Pipeline for bulk long read RNA-seq data processing (deprecated)*

Description

This function is deprecated. Use `BulkPipeline` instead.

Usage

```
bulk_long_pipeline(
  annotation,
  fastq,
  outdir,
  genome_fa,
  minimap2 = NULL,
  config_file
)
```

Arguments

`annotation`	The file path to the annotation file in GFF3 / GTF format.
`fastq`	Path to the FASTQ file or a directory containing FASTQ files. Each file will be processed as an individual sample.
`outdir`	Path to the output directory. If it does not exist, it will be created.
`genome_fa`	The file path to the reference genome in FASTA format.
`minimap2`	(optional) The path to the minimap2 binary. If not provided, FLAMES will use a copy from bioconda via `basilisk`.
`config_file`	Path to the JSON configuration file. See `create_config` for creating one.

Value

A SummarizedExperiment object containing the transcript counts.

See Also

BulkPipeline for the new pipeline function. SingleCellPipeline for single cell pipelines, MultiSampleSCPipeline for multi sample single cell pipelines.

Examples

```
outdir <- tempfile()
dir.create(outdir)
# simulate 3 samples via sampling
reads <- ShortRead::readFastq(
  system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES")
)
dir.create(file.path(outdir, "fastq"))
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample1.fq.gz"),
  mode = "w", full = FALSE
)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample2.fq.gz"),
  mode = "w", full = FALSE
)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads,
  file.path(outdir, "fastq/sample3.fq.gz"),
  mode = "w", full = FALSE
)
# prepare the reference genome
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
se <- bulk_long_pipeline(
  fastq = file.path(outdir, "fastq"),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  outdir = outdir, genome_fa = genome_fa,
  config_file = create_config(outdir, type = "sc_3end", threads = 1, no_flank = TRUE)
)
se
```

---

combine_sce *Combine SCE*

Description

Combine FLT-seq SingleCellExperiment objects

Usage

```
combine_sce(sce_with_lr, sce_without_lr)
```

Arguments

sce_with_lr	A SingleCellExperiment object with both long and short reads. The long-read transcript counts should be stored in the 'transcript' altExp slot.
sce_without_lr	A SingleCellExperiment object with only short reads.

Details

For protcols like FLT-seq that generate two libraries, one with both short and long reads, and one with only short reads, this function combines the two libraries into a single SingleCellExperiment object. For the library with both long and short reads, the long-read transcript counts should be stored in the 'transcript' altExp slot of the SingleCellExperiment object. This function will combine the short-read gene counts of both libraries, and for the transcripts counts, it will leave NA values for the cells from the short-read only library. The sc_impute_transcript function can then be used to impute the NA values.

Value

A SingleCellExperiment object with combined gene counts and a "transcript" altExp slot.

Examples

```
with_lr <- SingleCellExperiment::SingleCellExperiment(assays = list(counts = matrix(rpois(100, 5), ncol = 10)))
without_lr <- SingleCellExperiment::SingleCellExperiment(assays = list(counts = matrix(rpois(200, 5), ncol = 20)))
long_read <- SingleCellExperiment::SingleCellExperiment(assays = list(counts = matrix(rpois(50, 5), ncol = 10)))
SingleCellExperiment::altExp(with_lr, "transcript") <- long_read
SummarizedExperiment::colData(with_lr)$Barcode <- paste0(1:10, "-1")
SummarizedExperiment::colData(without_lr)$Barcode <- paste0(8:27, "-1")
rownames(with_lr) <- as.character(101:110)
rownames(without_lr) <- as.character(103:112)
rownames(long_read) <- as.character(1001:1005)
combined_sce <- FLAMES::combine_sce(sce_with_lr = with_lr, sce_without_lr = without_lr)
combined_sce
```

config *Get pipeline configurations*

Description

This function returns the configuration of the pipeline.

Usage

```
config(pipeline)

## S4 method for signature 'FLAMES.Pipeline'
config(pipeline)
```

Arguments

pipeline	An object of class 'FLAMES.Pipeline'.

Value

A list containing the configuration of the pipeline.

Examples

```
pipeline <- example_pipeline(type = "BulkPipeline")
config(pipeline)
```

config<- *Set pipeline configurations*

Description

This function sets the configuration of the pipeline.

Usage

```
config(pipeline) <- value

## S4 replacement method for signature 'FLAMES.Pipeline'
config(pipeline) <- value
```

Arguments

pipeline	An pipeline of class 'FLAMES.Pipeline'.
value	A list containing the configuration of the pipeline, or a path to a JSON configuration file.

Value

An pipeline of class 'FLAMES.Pipeline' with the updated configuration.

Examples

```
pipeline <- example_pipeline(type = "BulkPipeline")
# Set a new configuration
config(pipeline) <- create_config(outdir = tempdir())
```

`controllers` *Get controllers*

Description

Gets the controllers for the pipeline.

Usage

```
controllers(pipeline)

## S4 method for signature 'FLAMES.Pipeline'
controllers(pipeline)
```

Arguments

`pipeline`	A FLAMES.Pipeline object.

Value

A named list of `crew_class_controller` objects, where each controller corresponds to a step in the pipeline.

Examples

```
pipeline <- example_pipeline(type = "MultiSampleSCPipeline")
controllers(pipeline) # get the controllers
```

controllers<- *Set controllers*

Description

Sets the controllers for the pipeline.

Usage

```
controllers(pipeline) <- value

## S4 replacement method for signature 'FLAMES.Pipeline'
controllers(pipeline) <- value
```

Arguments

pipeline	A FLAMES.Pipeline object.
value	A crew_class_controller object or a named list of crew_class_controller objects. If a single controller is provided, it will be used for all steps in the pipeline. If a named list is provided, steps with names that match the names of the list will use the corresponding controller, and steps without a specified controller will use the current R session.

Value

An updated FLAMES.Pipeline object with the specified controllers.

Examples

```
pipeline <- example_pipeline()
# Only set the genome alignment controller
controllers(pipeline) <- list(genome_alignment = crew::crew_controller_local())
# Same as above
controllers(pipeline)[["genome_alignment"]] <- crew::crew_controller_local()
# Set a controller for all steps
controllers(pipeline) <- crew::crew_controller_local()
# Unset all controllers and use the current R session
controllers(pipeline) <- list()
```

convolution_filter *Convolution filter for smoothing transcript coverages*

Description

Filter out transcripts with sharp drops / rises in coverage, to be used in `filter_coverage` to remove transcripts with potential misalignments / internal priming etc. Filtering is done by convolving the coverage with a kernal of 1s and -1s (e.g. `c(1, 1, -1, -1)`, where the width of the 1s and -1s are determined by the `width` parameter), and check if the maximum absolute value of the convolution is below a threshold. If the convolution is below the threshold, `TRUE` is returned, otherwise `FALSE`.

Usage

```
convolution_filter(x, threshold = 0.15, width = 2, trim = 0.05)
```

Arguments

`x`	numeric vector of coverage values
`threshold`	numeric, the threshold for the maximum absolute value of the convolution
`width`	numeric, the width of the 1s and -1s in the kernal. E.g. `width = 2` will result in a kernal of `c(1, 1, -1, -1)`
`trim`	numeric, the proportion of the coverage values to ignore at both ends before convolution.

Value

logical, `TRUE` if the transcript passes the filter, `FALSE` otherwise

Examples

```
# A >30% drop in coverage will fail the filter with threshold = 0.3
convolution_filter(c(1, 1, 1, 0.69, 0.69, 0.69), threshold = 0.3)
convolution_filter(c(1, 1, 1, 0.71, 0.7, 0.7), threshold = 0.3)
```

create_config *Create Configuration File From Arguments*

Description

Create Configuration File From Arguments

Usage

```
create_config(outdir, type = "sc_3end", ...)
```

Arguments

`outdir`	the destination directory for the configuration file
`type`	use an example config, available values: **"sc_3end"** - config for 10x 3' end ONT reads **"SIRV"** - config for the SIRV example reads
`...`	Configuration parameters (using dot for nested parameters) **seed** - Integer. Seed for minimap2. **threads** - Number of threads to use. **do_barcode_demultiplex** - Boolean. Specifies whether to run the barcode demultiplexing step. **do_genome_alignment** - Boolean. Specifies whether to run the genome alignment step. `TRUE` is recommended **do_gene_quantification** - Boolean. Specifies whether to run gene quantification using the genome alignment results. `TRUE` is recommended **do_isoform_identification** - Boolean. Specifies whether to run the isoform identification step. `TRUE` is recommended **bambu_isoform_identification** - Boolean. Whether to use Bambu for isoform identification. **multithread_isoform_identification** - Boolean. Whether to use FLAMES' new multithreaded Cpp implementation for isoform identification. **do_read_realignment** - Boolean. Specifies whether to run the read realignment step. `TRUE` is recommended **do_transcript_quantification** - Boolean. Specifies whether to run the transcript quantification step. `TRUE` is recommended **barcode_parameters.max_bc_editdistance** - Maximum edit distance for barcode matching **barcode_parameters.pattern.primer** - Primer sequence pattern **isoform_parameters.max_dist** - Maximum distance allowed when merging splicing sites **...** - Other nested parameters, using dot to indicate nested section

Details

Create a list object containing the arguments supplied in a format usable for the FLAMES pipeline, and writes the object to a JSON file, which is located with the prefix 'config_' in the supplied `outdir`. Default values from `extdata/config_sclr_nanopore_3end.json` will be used for unprovided parameters.

Parameters can be specified using dot to indicate nested sections, e.g., `barcode_parameters.max_bc_editdistance = 3` or `barcode_parameters.pattern.primer = "ATCG"`. Alternatively, you can open the created config file and edit it manually.

Value

file path to the config file created

Examples

```
# create the default configuration file
outdir <- tempdir()
config <- create_config(outdir)

# create config with custom parameters including nested ones
config <- create_config(outdir,
  threads = 16,
  barcode_parameters.max_bc_editdistance = 3,
  barcode_parameters.pattern.primer = "ATCGATCG",
  isoform_parameters.min_sup_cnt = 10,
  # use the coverage model in oarfish
  # via supplying additional CLI arguments
  additional_arguments.oarfish = c("--model-coverage")
)
```

---

create_sce_from_dir *Create* SingleCellExperiment *object from* FLAMES *output folder*

Description

Create SingleCellExperiment object from FLAMES output folder

Usage

```
create_sce_from_dir(outdir, annotation, quantification = "FLAMES")
```

Arguments

outdir	The folder containing FLAMES output files
annotation	the annotation file that was used to produce the output files
quantification	(Optional) the quantification method used to generate the output files (either "FLAMES" or "Oarfish".). If not specified, the function will attempt to determine the quantification method.

Value

a list of SingleCellExperiment objects if multiple transcript matrices were found in the output folder, or a SingleCellExperiment object if only one were found

Examples

```
outdir <- tempfile()
dir.create(outdir)
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
```

```
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
annotation <- system.file("extdata", "rps24.gtf.gz", package = "FLAMES")

sce <- sc_long_pipeline(
  genome_fa = genome_fa,
  fastq = system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  annotation = annotation,
  outdir = outdir,
  barcodes_file = bc_allow,
  config_file = create_config(
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex",
    oarfish_quantification = FALSE
  )
)
sce_2 <- create_sce_from_dir(outdir, annotation)
```

`create_se_from_dir` *Create* `SummarizedExperiment` *object from* `FLAMES` *output folder*

Description

Create `SummarizedExperiment` object from `FLAMES` output folder

Usage

```
create_se_from_dir(outdir, annotation, quantification = "FLAMES")
```

Arguments

`outdir`	The folder containing `FLAMES` output files
`annotation`	(Optional) the annotation file that was used to produce the output files
`quantification`	(Optional) the quantification method used to generate the output files (either "FLAMES" or "Oarfish".). If not specified, the function will attempt to determine the quantification method.

Value

a `SummarizedExperiment` object

Examples

```
ppl <- example_pipeline("BulkPipeline")
ppl <- run_FLAMES(ppl)
se1 <- experiment(ppl)
se2 <- create_se_from_dir(ppl@outdir, ppl@annotation)
```

`create_spe` *Create a SpatialExperiment object*

Description

This function creates a SpatialExperiment object from a SingleCellExperiment object and a spatial barcode file.

Usage

```
create_spe(
  sce,
  spatial_barcode_file,
  mannual_align_json,
  image,
  tissue_positions_file
)
```

Arguments

`sce`	The SingleCellExperiment object obtained from running the `sc_long_pipeline` function.
`spatial_barcode_file`	The path to the spatial barcode file, e.g. `"spaceranger-2.1.1/lib/python/cellranger/barcodes/vis`
`mannual_align_json`	The path to the mannual alignment json file.
`image`	'DataFrame' containing the image data. See `?SpatialExperiment::readImgData` and `?SpatialExperiment::SpatialExperiment`.
`tissue_positions_file`	The path to Visium positions file, e.g. `"spaceranger-2.1.1/lib/python/cellranger/barcodes/visi`

Value

A SpatialExperiment object.

cutadapt *cutadapt wrapper*

Description

trim TSO adaptor with cutadapt

Usage

```
cutadapt(args)
```

Arguments

args	arguments to be passed to cutadapt

Value

Exit code of cutadapt

Examples

```
cutadapt("-h")
```

demultiplex_sockeye *Demultiplex reads using Sockeye outputs*

Description

Demultiplex reads using the `cell_umi_gene.tsv` file from Sockeye.

Usage

```
demultiplex_sockeye(fastq_dir, sockeye_tsv, out_fq)
```

Arguments

fastq_dir	The folder containing FASTQ files from Sockeye's output under `ingest/chunked_fastqs`.
sockeye_tsv	The `cell_umi_gene.tsv` file from Sockeye.
out_fq	The output FASTQ file.

Value

returns NULL

example_pipeline *Example pipelins*

Description

Provides example pipelines for bulk, single cell and multi-sample single cell.

Usage

```
example_pipeline(type = "SingleCellPipeline", outdir)
```

Arguments

type	The type of pipeline to create. Options are "SingleCellPipeline", "BulkPipeline", and "MultiSampleSCPipeline".
outdir	(Optional) The output directory where the example pipeline will be created. If not provided, a temporary directory will be created.

Value

A pipeline object of the specified type.

See Also

SingleCellPipeline for creating the single cell pipeline, BulkPipeline for bulk long data, MultiSampleSCPipeline for multi sample single cell pipelines.

Examples

```
example_pipeline("SingleCellPipeline")
```

experiment *Get pipeline results*

Description

This function returns the results of the pipeline as a SummarizedExperiment object, a SingleCellExperiment object, or a list of SingleCellExperiment objects, depending on the pipeline type.

Usage

```
experiment(pipeline)

## S4 method for signature 'FLAMES.Pipeline'
experiment(pipeline)
```

Arguments

pipeline	A FLAMES.Pipeline object.

Value

A SummarizedExperiment object, a SingleCellExperiment object, or a list of SingleCellExperiment objects.

Examples

```
pipeline <- example_pipeline(type = "BulkPipeline")
pipeline <- run_FLAMES(pipeline)
se <- experiment(pipeline)
```

---

fake_stranded_gff *Fake stranded GFF file*

Description

Check if all the transcript in the annotation is stranded. If not, convert to '+'.

Usage

```
fake_stranded_gff(gff_file)
```

Value

Path to the temporary file with unstranded transcripts converted to '+'.

---

filter_annotation *filter annotation for plotting coverages*

Description

Removes isoform annotations that could produce ambigious reads, such as isoforms that only differ by the 5' / 3' end. This could be useful for plotting average coverage plots.

Usage

```
filter_annotation(annotation, keep = "tss_differ")
```

Arguments

annotation	path to the GTF annotation file, or the parsed GenomicRanges object with a valid `transcript_id` column, and each Range representing a transcript.
keep	string, one of 'tss_differ' (only keep isoforms that all differ by the transcription start site position), 'tes_differ' (only keep those that differ by the transcription end site position), 'both' (only keep those that differ by both the start and end site), or 'single_transcripts' (only keep genes that contains a sinlge transcript).

Value

GenomicRanges of the filtered isoforms

Examples

```
filtered_annotation <- filter_annotation(
  system.file("extdata", "rps24.gtf.gz", package = 'FLAMES'), keep = 'tes_differ')
filtered_annotation
```

---

filter_coverage *Filter transcript coverage*

Description

Filter the transcript coverage by applying a filter function to the coverage values.

Usage

```
filter_coverage(x, filter_fn = convolution_filter)
```

Arguments

x	The tibble returned by get_coverage, or a BAM file path, or a GAlignments object.
filter_fn	The filter function to apply to the coverage values. The function should take a numeric vector of coverage values and return a logical value (TRUE if the transcript passes the filter, FALSE otherwise). The default filter function is convolution_filter, which filters out transcripts with sharp drops / rises in coverage.

Value

a tibble of the transcript information and coverages, with transcipts that pass the filter

Examples

```
ppl <- example_pipeline("BulkPipeline")
steps(ppl)["isoform_identification"] <- FALSE
ppl <- run_step(ppl, "read_realignment")
x <- get_coverage(ppl@transcriptome_bam[[1]])
nrow(x)
filter_coverage(x) |>
  nrow()
```

`find_barcode` *Match Cell Barcodes*

Description

demultiplex reads with flexiplex

Usage

```
find_barcode(
  fastq,
  barcodes_file,
  max_bc_editdistance = 2,
  max_flank_editdistance = 8,
  reads_out,
  stats_out,
  threads = 1,
 pattern = c(primer = "CTACACGACGCTCTTCCGATCT", BC = paste0(rep("N", 16), collapse =
  ""), UMI = paste0(rep("N", 12), collapse = ""), polyT = paste0(rep("T", 9), collapse
    = "")),
  TSO_seq = "",
  TSO_prime = 3,
  strand = "+",
  cutadapt_minimum_length = 1,
  full_length_only = FALSE
)
```

Arguments

`fastq`	A path to a FASTQ file or a directory containing FASTQ files.
`barcodes_file`	path to file containing barcode allow-list, with one barcode in each line
`max_bc_editdistance`	max edit distances for the barcode sequence
`max_flank_editdistance`	max edit distances for the flanking sequences (primer and polyT)
`reads_out`	path to output FASTQ file
`stats_out`	path of output stats file

threads	number of threads to be used
pattern	named character vector defining the barcode pattern
TSO_seq	TSO sequence to be trimmed
TSO_prime	either 3 (when TSO_seq is on 3' the end) or 5 (on 5' end)
strand	strand of the barcode pattern, either '+' or '-' (read will be reverse complemented after barcode matching if '-')
cutadapt_minimum_length	minimum read length after TSO trimming (cutadapt's –minimum-length)
full_length_only	boolean, when TSO sequence is provided, whether reads without TSO are to be discarded

Details

This function demultiplexes reads by searching for flanking sequences (adaptors) around the barcode sequence, and then matching against allowed barcodes.

Value

a list containing: reads_tb (tibble of read demultiplexed information) and input, output, read1_with_adapter from cutadapt report (if TSO trimming is performed)

Examples

```
outdir <- tempfile()
dir.create(outdir)
bc_allow <- file.path(outdir, "bc_allow.tsv")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
find_barcode(
  fastq = system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  stats_out = file.path(outdir, "bc_stat.tsv.gz"),
  reads_out = file.path(outdir, "demultiplexed.fastq.gz"),
  barcodes_file = bc_allow,
  TSO_seq = "AAGCAGTGGTATCAACGCAGAGTACATGGG", TSO_prime = 5,
  strand = '-', cutadapt_minimum_length = 10, full_length_only = TRUE
)
```

---

find_bin *Find path to a binary Wrapper for Sys.which to find path to a binary*

Description

This function is a wrapper for base::Sys.which to find the path to a command. It also searches within the FLAMES basilisk conda environment. This function also replaces "" with NA in the output of base::Sys.which to make it easier to check if the binary is found.

Usage

```
find_bin(command)
```

Arguments

command	character, the command to search for

Value

character, the path to the command or NA

Examples

```
find_bin("minimap2")
```

---

find_isoform *Isoform identification*

Description

Long-read isoform identification with FLAMES or bambu.

Usage

```
find_isoform(annotation, genome_fa, genome_bam, outdir, config)
```

Arguments

annotation	Path to annotation file. If configured to use bambu, the annotation must be provided as GTF file.
genome_fa	The file path to genome fasta file.
genome_bam	File path to BAM alignment file. Multiple files could be provided.
outdir	The path to directory to store all output files.
config	Parsed FLAMES configurations.

Value

The updated annotation and the transcriptome assembly will be saved in the output folder as `isoform_annotated.gff3` (GTF if bambu is selected) and `transcript_assembly.fa` respectively.

`find_variants` *bulk variant identification*

Description

Treat each bam file as a bulk sample and identify variants against the reference

Usage

```
find_variants(
  bam_path,
  reference,
  annotation,
  min_nucleotide_depth = 100,
  homopolymer_window = 3,
  annotated_region_only = FALSE,
  names_from = "gene_name",
  threads = 1
)
```

Arguments

`bam_path`	character(1) or character(n): path to the bam file(s) aligned to the reference genome (NOT the transcriptome!).
`reference`	DNAStringSet: the reference genome
`annotation`	GRanges: the annotation of the reference genome. You can load a GTF/GFF annotation file with `anno <- rtracklayer::import(file)`.
`min_nucleotide_depth`	integer(1): minimum read depth for a position to be considered a variant.
`homopolymer_window`	integer(1): the window size to calculate the homopolymer percentage. The homopolymer percentage is calculated as the percentage of the most frequent nucleotide in a window of `-homopolymer_window` to `homopolymer_window` nucleotides around the variant position, excluding the variant position itself. Calculation of the homopolymer percentage is skipped when `homopolymer_window = 0`. This is useful for filtering out Nanopore sequencing errors in homopolymer regions.
`annotated_region_only`	logical(1): whether to only consider variants outside annotated regions. If `TRUE`, only variants outside annotated regions will be returned. If `FALSE`, all variants will be returned, which could take significantly longer time.
`names_from`	character(1): the column name in the metadata column of the annotation (`mcols(annotation)[, names_from]`) to use for the `region` column in the output.
`threads`	integer(1): number of threads to use. Threading is done over each annotated region and (if `annotated_region_only = FALSE`) unannotated gaps for each bam file.

Details

Each bam file is treated as a bulk sample to perform pileup and identify variants. You can run `sc_mutations` with the variants identified with this function to get single-cell allele counts. Note that reference genome FASTA files may have the chromosome names field as '>chr1 1' instead of '>chr1'. You may need to remove the trailing number to match the chromosome names in the bam file, for example with `names(ref) <- sapply(names(ref), function(x) strsplit(x, " ")[[1]][1])`.

Value

A tibble with columns: seqnames, pos, nucleotide, count, sum, freq, ref, region, homopolymer_pct, bam_path The homopolymer percentage is calculated as the percentage of the most frequent nucleotide in a window of `homopolymer_window` nucleotides around the variant position, excluding the variant position itself.

Examples

```
ppl <- example_pipeline("SingleCellPipeline")
ppl <- run_step(ppl, "genome_alignment")
variants <- find_variants(
  bam_path = ppl@genome_bam,
  reference = ppl@genome_fa,
  annotation = ppl@annotation,
  min_nucleotide_depth = 4
)
head(variants)
```

`FLAMES` *FLAMES: full-length analysis of mutations and splicing*

Description

FLAMES: full-length analysis of mutations and splicing

Value

invisible()

`flexiplex` *Rcpp port of flexiplex*

Description

demultiplex reads with flexiplex, for detailed description, see documentation for the original flexiplex: https://davidsongroup.github.io/flexiplex

Usage

```
flexiplex(
  reads_in,
  barcodes_file,
  bc_as_readid,
  max_bc_editdistance,
  max_flank_editdistance,
  pattern,
  reads_out,
  stats_out,
  bc_out,
  reverseCompliment,
  n_threads
)
```

Arguments

`reads_in`	Input FASTQ or FASTA file
`barcodes_file`	barcode allow-list file
`bc_as_readid`	bool, whether to add the demultiplexed barcode to the read ID field
`max_bc_editdistance`	max edit distance for barcode '
`max_flank_editdistance`	max edit distance for the flanking sequences '
`pattern`	StringVector defining the barcode structure, see [find_barcode]
`reads_out`	output file for demultiplexed reads
`stats_out`	output file for demultiplexed stats
`bc_out`	WIP
`reverseCompliment`	bool, whether to reverse complement the reads after demultiplexing
`n_threads`	number of threads to be used during demultiplexing

Value

integer return value. 0 represents normal return.

`get_coverage` *Get read coverages from BAM file*

Description

Get the read coverages for each transcript in the BAM file (or a GAlignments object). The read coverages are sampled at 100 positions along the transcript, and the coverage is scaled by dividing the coverage at each position by the total read counts for the transcript. If a BAM file is provided, alignment with MAPQ < 5, secondary alignments and supplementary alignments are filtered out. A `GAlignments` object can also be provided in case alternative filtering is desired.

Usage

```
get_coverage(bam, min_counts = 10, remove_UTR = FALSE, annotation)
```

Arguments

`bam`	path to the BAM file, or a parsed GAlignments object
`min_counts`	numeric, the minimum number of alignments required for a transcript to be included
`remove_UTR`	logical, if `TRUE`, remove the UTRs from the coverage
`annotation`	(Required if `remove_UTR = TRUE`) path to the GTF annotation file

Value

a tibble of the transcript information and coverages, with the following columns:

- `transcript`: the transcript name / ID
- `read_counts`: the total number of aligments for the transcript
- `coverage_1-100`: the coverage at each of the 100 positions along the transcript
- `tr_length`: the length of the transcript

Examples

```
ppl <- example_pipeline("BulkPipeline")
steps(ppl)["isoform_identification"] <- FALSE
ppl <- run_step(ppl, "read_realignment")
x <- get_coverage(ppl@transcriptome_bam[[1]])
head(x)
```

get_GRangesList *Parse FLAMES' GFF output*

Description

Parse FLAMES' GFF ouputs into a Genomic Ranges List

Usage

```
get_GRangesList(
  file,
  feature.type = c("exon", "utr"),
  drop.cols = c("type", "exon_number", "exon_id", "level", "Parent")
)
```

Arguments

file	the GFF file to parse
feature.type	The type of features to extract from the GFF file. Default is c("exon", "utr").
drop.cols	Columns to drop from the metadata. Default is c("type", "exon_number", "exon_id", "level"), which are exon-specific metadata that may not be relevant when keeping just the first row (exon).

Value

A list containing a GRangesList of isoforms and a DataFrame, which have the same number of rows as the number of unique transcript IDs in the GFF file.

gff2bed *Convert GFF/GTF to BED file*

Description

Convert GFF/GTF to BED file

Usage

```
gff2bed(gff, bed)
```

Arguments

gff	Path to the GFF/GTF file
bed	Path to the output BED file to be written

Value

invisible, the BED file is written to the specified path

index_genome *Index the reference genome for minimap2*

Description

Calls minimap2 to index the reference genome.

Usage

```
index_genome(pipeline, path, additional_args = c("-k", "14"))

## S4 method for signature 'FLAMES.Pipeline'
index_genome(pipeline, path, additional_args = c("-k", "14"))
```

Arguments

pipeline	A FLAMES.Pipeline object.
path	The file path to save the minimap2 index. If not provided, it will be saved to the output directory with the name "genome.mmi".
additional_args	(optional) Additional arguments to pass to minimap2.

Value

A SummarizedExperiment object, a SingleCellExperiment object, or a list of SingleCellExperiment objects.

Examples

```
pipeline <- example_pipeline(type = "BulkPipeline")
pipeline <- index_genome(pipeline)
```

load_config *Load Configurations*

Description

Loads a configuration file and fills in missing values with defaults from the package's default configuration.

Usage

```
load_config(config_file, type = "sc_3end")
```

Arguments

config_file	Path to the configuration JSON file
type	Config type to use for defaults ("sc_3end" or "SIRV")

Value

A complete configuration list with all parameters filled

---

merge_configs_recursive

Recursively Merge Configuration Lists

Description

Internal function to recursively merge configuration lists, filling missing values from defaults while preserving user values

Usage

```
merge_configs_recursive(default_config, user_config)
```

Arguments

default_config	Default configuration list
user_config	User configuration list

Value

Merged configuration list

Note

Special case: when user_config contains barcode_parameters.pattern as a list, the entire pattern list is preserved as-is without merging with defaults to maintain user-specified order and structure.

`minimap2_align` *Minimap2 Align to Genome*

Description

Uses minimap2 to align sequences agains a reference databse. Uses options '-ax splice -t 12 -k14 –secondary=no `fa_file` `fq_in`'

Usage

```
minimap2_align(
  fq_in,
  fa_file,
  config,
  outfile,
  minimap2_args,
  sort_by,
  minimap2,
  samtools,
  threads = 1,
  tmpdir
)
```

Arguments

`fq_in`	File path to the fastq file used as a query sequence file
`fa_file`	Path to the fasta file used as a reference database for alignment
`config`	Parsed list of FLAMES config file
`outfile`	Path to the output file
`minimap2_args`	Arguments to pass to minimap2, see minimap2 documentation for details.
`sort_by`	Column to sort the bam file by, see `samtools sort` for details
`minimap2`	Path to minimap2 binary
`samtools`	path to the samtools binary.
`threads`	Integer, threads for minimap2 to use, see minimap2 documentation for details,
`tmpdir`	Temporary directory to use for intermediate files. FLAMES will try to detect cores if this parameter is not provided.

Value

a `data.frame` summarising the reads aligned

`MultiSampleSCPipeline` *Pipeline for multi-sample long-read scRNA-seq data*

Description

Semi-supervised isofrom detection and annotation for long read data. This variant is meant for multi-sample scRNA-seq data. Specific parameters can be configured in the config file (see `create_config`), input files are specified via arguments.

Usage

```
MultiSampleSCPipeline(
  config_file,
  outdir,
  fastq,
  annotation,
  genome_fa,
  genome_mmi,
  minimap2,
  samtools,
  barcodes_file,
  expect_cell_number,
  controllers
)
```

Arguments

`config_file`	Path to the JSON configuration file. See `create_config` for creating one.
`outdir`	Path to the output directory. If it does not exist, it will be created.
`fastq`	A named vector of fastq file (or folder) paths. Each element of the vector will be treated as a sample. The names of the vector will be used as the sample names. If not named, the sample names will be generated from the file names.
`annotation`	The file path to the annotation file in GFF3 / GTF format.
`genome_fa`	The file path to the reference genome in FASTA format.
`genome_mmi`	(optional) The file path to minimap2's index reference genome.
`minimap2`	(optional) The path to the minimap2 binary. If not provided, FLAMES will use a copy from bioconda via `basilisk`.
`samtools`	(optional) The path to the samtools binary. If not provided, FLAMES will use a copy from bioconda via `basilisk`.
`barcodes_file`	The file with expected cell barcodes, with each barcode on a new line.
`expect_cell_number`	The expected number of cells in the sample. This is used if `barcodes_file` is not provided. See `BLAZE` for more details.
`controllers`	(optional, **experimental**) A `crew_class_controller` object for running certain steps

Details

By default the pipeline starts with demultiplexing the input fastq data. If the cell barcodes are known apriori (e.g. via coupled short-read sequencing), the `barcodes_file` argument can be used to specify a file containing the cell barcodes, and a modified Rcpp version of `flexiplex` will be used; otherwise, `expect_cell_number` need to be provided, and `BLAZE` will be used to generate the cell barcodes. The pipeline then aligns the reads to the genome using `minimap2`. The alignment is then used for isoform detection (either using `FLAMES` or `bambu`, can be configured). The reads are then realigned to the detected isoforms. Finally, a transcript count matrix is generated (either using `FLAMES`'s simplistic counting or `oarfish`'s Expectation Maximization algorithm, can be configured). The results can be accssed with `experiment(pipeline)`. If the pipeline errored out / new steps were configured, it can be resumed by calling `resume_FLAMES(pipeline)`

Value

A `FLAMES.MultiSampleSCPipeline` object. The pipeline can be run using the `run_FLAMES` function. The resulting list of SingleCellExperiment objects can be accessed using the `experiment` method.

See Also

`SingleCellPipeline` for single-sample long data and more details on the pipeline output, `create_config` for creating a configuration file, `BulkPipeline` for bulk long data.

Examples

```
reads <- ShortRead::readFastq(
  system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES")
)
outdir <- tempfile()
dir.create(outdir)
dir.create(file.path(outdir, "fastq"))
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample1.fq.gz"), mode = "w", full = FALSE)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample2.fq.gz"), mode = "w", full = FALSE)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads,
  file.path(outdir, "fastq/sample3.fq.gz"), mode = "w", full = FALSE)
ppl <- MultiSampleSCPipeline(
  config_file = create_config(
```

```
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex",
    pipeline_parameters.threads = 1,
    alignment_parameters.no_flank = TRUE
  ),
  outdir = outdir,
  fastq = c("sampleA" = file.path(outdir, "fastq"),
    "sample1" = file.path(outdir, "fastq", "sample1.fq.gz"),
    "sample2" = file.path(outdir, "fastq", "sample2.fq.gz"),
    "sample3" = file.path(outdir, "fastq", "sample3.fq.gz")),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  genome_fa = genome_fa,
  barcodes_file = rep(bc_allow, 4)
)
ppl <- run_FLAMES(ppl)
experiment(ppl)
```

`mutation_positions` *Calculate mutation positions within the gene body*

Description

Given a set of mutations and gene annotation, calculate the position of each mutation within the gene body they are in.

Usage

```
mutation_positions(
  mutations,
  annotation,
  type = "relative",
  bin = FALSE,
  by = c(region = "gene_name"),
  threads = 1
)
```

Arguments

`mutations`	either the tibble output from `find_variants`. It must have columns `seqnames`, `pos`, and a third column for specifying the gene id or gene name. The mutation must be within the gene region.
`annotation`	Either path to the annotation file (GTF/GFF) or a GRanges object of the gene annotation.
`type`	character(1): the type of position to calculate. Can be one of "TSS" (distance from the transcription start site), "TES" (distance from the transcription end site), or "relative" (relative position within the gene body).
`bin`	logical(1): whether to bin the relative positions into 100 bins. Only applicable when `type = "relative"`.

by	character(1): the column name in the annotation to match with the gene annotation. E.g. c("region" = "gene_name") to match the 'region' column in the mutations with the 'gene_name' column in the annotation.
threads	integer(1): number of threads to use.

Value

A numeric vector of positions of each mutation within the gene body. When type = "relative", the positions are normalized to the gene length, ranging from 0 (start of the gene) to 1 (end of the gene). When type = "TSS" / type = "TES", the distances from the transcription start / end site. If bin = TRUE, and type = "relative", the relative positions are binned into 100 bins along the gene body, and the output is a matrix with the number of mutations in each bin, the rows are named by the by column (e.g. gene name).

Examples

```
variants <- data.frame(
  seqnames = rep("chr14", 8),
  pos = c(1084, 1085, 1217, 1384, 2724, 2789, 5083, 5147),
  region = rep("Rps24", 8)
)
positions <-
 mutation_positions(
   mutations = variants,
   annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES")
 )
```

---

mutation_positions_single

mutation positions within the gene body

Description

Given a set of mutations and a gene annotation, calculate the position of each mutation within the gene body. The gene annotation must have the following types: "gene" and "exon". The gene annotation must be for one gene only. The mutations must be within the gene region. The function will merge overlapping exons and calculate the position of each mutation within the gene body, excluding intronic regions.

Usage

```
mutation_positions_single(mutations, annotation_grange, type, verbose = TRUE)
```

Arguments

mutations	either the tibble output from find_variants or a GRanges object. Make sure to filter it for only the gene of interest.

annotation_grange
: GRanges: the gene annotation. Must have the following types: "gene" and "exon".

type
: character(1): the type of position to calculate. Can be one of "TSS" (distance from the transcription start site), "TES" (distance from the transcription end site), or "relative" (relative position within the gene body).

verbose
: logical(1): whether to print messages.

Value

A numeric vector of positions of each mutation within the gene body. When `type = "relative"`, the positions are normalized to the gene length, ranging from 0 (start of the gene) to 1 (end of the gene). When `type = "TSS"` / `type = "TES"`, the distances from the transcription start / end site.

---

plot_coverage *plot read coverages*

Description

Plot the average read coverages for each length bin or a perticular isoform

Usage

```
plot_coverage(
  x,
  quantiles = c(0, 0.2375, 0.475, 0.7125, 0.95, 1),
  length_bins = c(0, 1, 2, 5, 10, Inf),
  weight_fn = weight_transcripts,
  filter_fn,
  detailed = FALSE
)
```

Arguments

x
: path to the BAM file (aligning reads to the transcriptome), or the (GenomicAlignments::readGAlignments) parsed GAlignments object, or the tibble returned by `get_coverage`, or the filtered tibble returned by `filter_coverage`.

quantiles
: numeric vector to specify the quantiles to bin the transcripts lengths by if length_bins is missing. The length bins will be determined such that the read counts are distributed acording to the quantiles.

length_bins
: numeric vector to specify the sizes to bin the transcripts by

weight_fn
: function to calculate the weights for the transcripts. The function should take a numeric vector of read counts and return a numeric vector of weights. The default function is `weight_transcripts`, you can change its default parameters by passing an anonymous function like `function(x) weight_transcripts(x, type = 'equal')`.

filter_fn	Optional filter function to filter the transcripts before plotting. See the filter_fn parameter in filter_coverage for more details. Providing a filter fucntion here is the same as providing it in filter_coverage and then passing the result to this function.
detailed	logical, if TRUE, also plot the top 10 transcripts with the highest read counts for each length bin.

Value

a ggplot2 object of the coverage plot(s)

Examples

```
ppl <- example_pipeline("BulkPipeline")
steps(ppl)["isoform_identification"] <- FALSE
ppl <- run_step(ppl, "read_realignment")
# Plot the coverages directly from the BAM file
plot_coverage(ppl@transcriptome_bam[[1]])

# Get the coverage information first
coverage <- get_coverage(ppl@transcriptome_bam[[1]]) |>
  dplyr::filter(read_counts > 2) |> # Filter out transcripts with read counts < 3
 filter_coverage(filter_fn = convolution_filter) # Filter out transcripts with sharp drops / rises
# Plot the filtered coverages
plot_coverage(coverage, detailed = TRUE)
# filtering function can also be passed directly to plot_coverage
plot_coverage(ppl@transcriptome_bam[[1]], filter_fn = convolution_filter)
```

---

plot_demultiplex *Plot Cell Barcode demultiplex statistics*

Description

produce a barplot of cell barcode demultiplex statistics

Usage

```
plot_demultiplex(pipeline)

## S4 method for signature 'FLAMES.SingleCellPipeline'
plot_demultiplex(pipeline)
```

Arguments

pipeline	A FLAMES.SingleCellPipeline object

Value

a list of ggplot objects:

- reads_count_plot: stacked barplot of: demultiplexed reads
- knee_plot: knee plot of UMI counts before TSO trimming
- flank_editdistance_plot: flanking sequence (adaptor) edit-distance plot
- barcode_editdistance_plot: barcode edit-distance plot
- cutadapt_plot: if TSO trimming is performed, number of reads kept by cutadapt

Examples

```
pipeline <- example_pipeline("MultiSampleSCPipeline") |>
  run_step("barcode_demultiplex")
plot_demultiplex(pipeline)
```

---

plot_demultiplex_raw *Plot Cell Barcode demultiplex statistics*

Description

produce a barplot of cell barcode demultiplex statistics

Usage

```
plot_demultiplex_raw(find_barcode_result)
```

Arguments

`find_barcode_result`
: output from `find_barcode`

Value

a list of ggplot objects:

- reads_count_plot: stacked barplot of: demultiplexed reads
- knee_plot: knee plot of UMI counts before TSO trimming
- flank_editdistance_plot: flanking sequence (adaptor) edit-distance plot
- barcode_editdistance_plot: barcode edit-distance plot
- cutadapt_plot: if TSO trimming is performed, number of reads kept by cutadapt

Examples

```
outdir <- tempfile()
dir.create(outdir)
fastq_dir <- tempfile()
dir.create(fastq_dir)
file.copy(system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  file.path(fastq_dir, "musc_rps24.fastq.gz"))
sampled_lines <- readLines(file.path(fastq_dir, "musc_rps24.fastq.gz"), n = 400)
writeLines(sampled_lines, file.path(fastq_dir, "copy.fastq"))
bc_allow <- file.path(outdir, "bc_allow.tsv")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
find_barcode(
  fastq = fastq_dir,
  stats_out = file.path(outdir, "bc_stat.tsv.gz"),
  reads_out = file.path(outdir, "demultiplexed.fq"),
  barcodes_file = bc_allow, TSO_seq = "CCCATGTACTCTGCGTTGATACCACTGCTT"
) |>
  plot_demultiplex_raw()
```

---

plot_durations *Plot pipeline step durations*

Description

This function creates a horizontal bar plot showing the duration of each pipeline step using ggplot2.

Usage

```
plot_durations(x)

## S4 method for signature 'FLAMES.Pipeline'
plot_durations(x)
```

Arguments

x	A FLAMES.Pipeline object.

Value

A ggplot2 object.

Examples

```
pipeline <- example_pipeline("BulkPipeline")
pipeline <- run_FLAMES(pipeline)
plot_durations(pipeline)
```

plot_isoforms *Plot isoforms*

Description

Plot isoforms, either from a gene or a list of transcript ids.

Usage

```
plot_isoforms(
  sce,
  gene_id,
  transcript_ids,
  n = 4,
  format = "plot_grid",
  colors
)
```

Arguments

sce	The SingleCellExperiment object containing transcript counts, rowRanges and rowData with gene_id and transcript_id columns.
gene_id	The gene symbol of interest, ignored if transcript_ids is provided.
transcript_ids	The transcript ids to plot.
n	The number of top isoforms to plot from the gene. Ignored if transcript_ids is provided.
format	The format of the output, either "plot_grid" or "list".
colors	A character vector of colors to use for the isoforms. If not provided, gray will be used. for all isoforms.

Details

This function takes a SingleCellExperiment object and plots the top isoforms of a gene, or a list of specified transcript ids. Either as a list of plots or together in a grid. This function wraps the ggbio::geom_alignment function to plot the isoforms, and orders the isoforms by expression levels (when specifying a gene) or by the order of the transcript_ids.

Value

When format = "list", a list of ggplot objects is returned. Otherwise, a grid of the plots is returned.

Examples

```
data(scmixology_lib10_transcripts)
plot_isoforms(scmixology_lib10_transcripts, gene_id = "ENSG00000108107")
```

`plot_isoform_heatmap` *FLAMES heetmap plots*

Description

Plot expression heatmap of top n isoforms of a gene

Usage

```
plot_isoform_heatmap(
  sce,
  gene_id,
  transcript_ids,
  n = 4,
  isoform_legend_width = 7,
  col_low = "#313695",
  col_mid = "#FFFFBF",
  col_high = "#A50026",
  color_quantile = 1,
  cluster_palette,
  ...
)
```

Arguments

`sce`	The `SingleCellExperiment` object containing transcript counts, `rowRanges` and `rowData` with `gene_id` and `transcript_id` columns.
`gene_id`	The gene symbol of interest, ignored if `transcript_ids` is provided.
`transcript_ids`	The transcript ids to plot.
`n`	The number of top isoforms to plot from the gene. Ignored if `transcript_ids` is provided.
`isoform_legend_width`	The width of isoform legends in heatmaps, in `cm`.
`col_low`	Color for cells with low expression levels in UMAPs.
`col_mid`	Color for cells with intermediate expression levels in UMAPs.
`col_high`	Color for cells with high expression levels in UMAPs.
`color_quantile`	The lower and upper expression quantile to be displayed bewteen `col_low` and `col_high`, e.g. with `color_quantile = 0.95`, cells with expressions higher than 95% of other cells will all be shown in `col_high`, and cells with expression lower than 95% of other cells will all be shown in `col_low`.
`cluster_palette`	Optional, named vector of colors for the cluster annotations.
`...`	Additional arguments to pass to `Heatmap`.

Details

Takes `SingleCellExperiment` object and plots an expression heatmap with the isoform visualizations along genomic coordinates.

Value

a `ComplexHeatmap`

Examples

```
data(scmixology_lib10_transcripts)
scmixology_lib10_transcripts |>
  scuttle::logNormCounts() |>
  plot_isoform_heatmap(gene = "ENSG00000108107")
```

---

plot_isoform_reduced_dim

FLAMES isoform reduced dimensions plots

Description

Plot expression of top n isoforms of a gene in reduced dimensions

Usage

```
plot_isoform_reduced_dim(
  sce,
  gene_id,
  transcript_ids,
  n = 4,
  reduced_dim_name = "UMAP",
  use_gene_dimred = FALSE,
  expr_func = function(x) {
     SingleCellExperiment::logcounts(x)
 },
  col_low = "#313695",
  col_mid = "#FFFFBF",
  col_high = "#A50026",
  alpha = 0.5,
  size = 0.2,
  ggtheme = theme_minimal() + theme(axis.text = element_blank()),
  color_quantile = 1,
  format = "plot_grid",
  ...
)
```

Arguments

sce	The SingleCellExperiment object containing transcript counts, rowRanges and rowData with gene_id and transcript_id columns.
gene_id	The gene symbol of interest, ignored if transcript_ids is provided.
transcript_ids	The transcript ids to plot.
n	The number of top isoforms to plot from the gene. Ignored if transcript_ids is provided.
reduced_dim_name	The name of the reduced dimension to use for plotting cells.
use_gene_dimred	Whether to use gene-level reduced dimensions for plotting. Set to TRUE if the SingleCellExperiment has gene counts in main assay and transcript counts in altExp.
expr_func	The function to extract expression values from the SingleCellExperiment object. Default is logcounts. Alternatively, counts can be used for raw counts.
col_low	Color for cells with low expression levels in UMAPs.
col_mid	Color for cells with intermediate expression levels in UMAPs.
col_high	Color for cells with high expression levels in UMAPs.
alpha	The transparency of the points in the UMAPs.
size	The size of the points in the UMAPs.
ggtheme	The theme to use for the UMAPs.
color_quantile	The lower and upper expression quantile to be displayed bewteen col_low and col_high, e.g. with color_quantile = 0.95, cells with expressions higher than 95% of other cells will all be shown in col_high, and cells with expression lower than 95% of other cells will all be shown in col_low.
format	The format of the output, either "plot_grid" or "list".
...	Additional arguments to pass to plot_grid.

Details

Takes SingleCellExperiment object and plots an expression on reduced dimensions with the isoform visualizations along genomic coordinates.

Value

a ggplot object of the UMAP(s)

Examples

```
data(scmixology_lib10_transcripts, scmixology_lib10, scmixology_lib90)
scmixology_lib10 <-
  scmixology_lib10[, colSums(SingleCellExperiment::counts(scmixology_lib10)) > 0]
sce_lr <- scmixology_lib10[, colnames(scmixology_lib10) %in% colnames(scmixology_lib10_transcripts)]
SingleCellExperiment::altExp(sce_lr, "transcript") <-
  scmixology_lib10_transcripts[, colnames(sce_lr)]
```

```
combined_sce <- combine_sce(sce_lr, scmixology_lib90)
combined_sce <- combined_sce |>
  scuttle::logNormCounts() |>
  scater::runPCA() |>
  scater::runUMAP()
combined_imputed_sce <- sc_impute_transcript(combined_sce)
plot_isoform_reduced_dim(combined_sce, 'ENSG00000108107')
plot_isoform_reduced_dim(combined_imputed_sce, 'ENSG00000108107')
```

`plot_spatial_feature` *Plot feature on spatial image*

Description

This function plots a spatial point plot for given feature

Usage

```
plot_spatial_feature(
  spe,
  feature,
  opacity = 50,
  grayscale = TRUE,
  size = 1,
  assay_type = "counts",
  color = "red",
  ...
)
```

Arguments

`spe`	The SpatialExperiment object.
`feature`	The feature to plot. Could be either a feature name or index present in the assay or a numeric vector of length nrow(spe).
`opacity`	The opacity of the background tissue image.
`grayscale`	Whether to convert the background image to grayscale.
`size`	The size of the points.
`assay_type`	The assay that contains the given features. E.g. 'counts', 'logcounts'.
`color`	The maximum color for the feature. Minimum color is transparent.
`...`	Additional arguments to pass to `geom_point`.

Value

A ggplot object.

plot_spatial_isoform *Plot spatial pie chart of isoforms*

Description

This function plots a spatial pie chart for given features.

Usage

```
plot_spatial_isoform(spe, isoforms, assay_type = "counts", color_palette, ...)
```

Arguments

spe	The SpatialExperiment object.
isoforms	The isoforms to plot.
assay_type	The assay that contains the given features. E.g. 'counts', 'logcounts'.
color_palette	Named vector of colors for each isoform.
...	Additional arguments to pass to plot_spatial_pie, including opacity, grayscale, pie_scale.

Value

A ggplot object.

plot_spatial_pie *Plot spatial pie chart*

Description

This function plots a spatial pie chart for given features.

Usage

```
plot_spatial_pie(
  spe,
  features,
  assay_type = "counts",
  color_palette,
  opacity = 50,
  grayscale = TRUE,
  pie_scale = 0.8
)
```

Arguments

spe	The SpatialExperiment object.
features	The features to plot.
assay_type	The assay that contains the given features.
color_palette	Named vector of colors for each feature.
opacity	The opacity of the background tissue image.
grayscale	Whether to convert the background image to grayscale.
pie_scale	The size of the pie charts.

Value

A ggplot object.

---

quantify_gene *Gene quantification*

Description

Calculate the per gene UMI count matrix by parsing the genome alignment file.

Usage

```
quantify_gene(
  annotation,
  outdir,
  pipeline = "sc_single_sample",
  infq,
  in_bam,
  out_fastq,
  n_process,
  saturation_curve = TRUE,
  sample_names = NULL,
  random_seed = 2024
)
```

Arguments

annotation	The file path to the annotation file in GFF3 format
outdir	The path to directory to store all output files.
pipeline	The pipeline type as a character string, either `sc_single_sample` (single-cell, single-sample), `bulk` (bulk, single or multi-sample), or `sc_multi_sample` (single-cell, multiple samples)
infq	The input FASTQ file.
in_bam	The input BAM file(s) from the genome alignment step.

out_fastq	The output FASTQ file(s) to store deduplicated reads.
n_process	The number of processes to use for parallelization.
saturation_curve	Logical, whether to generate a saturation curve figure.
sample_names	A vector of sample names, default to the file names of input fastq files, or folder names if fastqs is a vector of folders.
random_seed	The random seed for reproducibility.

Details

After the genome alignment step (do_genome_align), the alignment file will be parsed to generate the per gene UMI count matrix. For each gene in the annotation file, the number of reads overlapping with the gene's genomic coordinates will be assigned to that gene. If a read overlaps multiple genes, it will be assigned to the gene with the highest number of overlapping nucleotides. If exon coordinates are included in the provided annotation, the decision will first consider the number of nucleotides aligned to the exons of each gene. In cases of a tie, the overlap with introns will be used as a tiebreaker. If there is still a tie after considering both exons and introns, a random gene will be selected from the tied candidates.

After the read-to-gene assignment, the per gene UMI count matrix will be generated. Specifically, for each gene, the reads with similar mapping coordinates of transcript termination sites (TTS, i.e. the end of the the read with a polyT or polyA) will be grouped together. UMIs of reads in the same group will be collapsed to generate the UMI counts for each gene.

Finally, a new fastq file with deduplicated reads by keeping the longest read in each UMI.

Value

The count matrix will be saved in the output folder as transcript_count.csv.gz.

---

quantify_transcript *Transcript quantification*

Description

Calculate the transcript count matrix by parsing the re-alignment file.

Usage

```
quantify_transcript(
  annotation,
  outdir,
  config,
  pipeline = "sc_single_sample",
  ...
)
```

Arguments

annotation	The file path to the annotation file in GFF3 format
outdir	The path to directory to store all output files.
config	Parsed FLAMES configurations.
pipeline	The pipeline type as a character string, either sc_single_sample (single-cell, single-sample),
...	Supply sample names as character vector (e.g. samples = c("name1", "name2", ...)) for muti-sample or bulk pipeline. bulk (bulk, single or multi-sample), or sc_multi_sample (single-cell, multiple samples)

Value

A SingleCellExperiment object for single-cell pipeline, a list of SingleCellExperiment objects for multi-sample pipeline, or a SummarizedExperiment object for bulk pipeline.

---

quantify_transcript_flames

FLAMES Transcript quantification

Description

Calculate the transcript count matrix by parsing the re-alignment file.

Usage

```
quantify_transcript_flames(
  annotation,
  outdir,
  config,
  pipeline = "sc_single_sample",
  samples
)
```

Arguments

annotation	The file path to the annotation file in GFF3 format
outdir	The path to directory to store all output files.
config	Parsed FLAMES configurations.
pipeline	The pipeline type as a character string, either sc_single_sample (single-cell, single-sample),
samples	A vector of sample names, required for sc_multi_sample pipeline. bulk (bulk, single or multi-sample), or sc_multi_sample (single-cell, multiple samples)

Value

A SingleCellExperiment object for single-cell pipeline, a list of SingleCellExperiment objects for multi-sample pipeline, or a SummarizedExperiment object for bulk pipeline.

---

resume_FLAMES *Resume a FLAMES pipeline*

Description

This function resumes a FLAMES pipeline by running configured but unfinished steps.

Usage

```
resume_FLAMES(pipeline)

## S4 method for signature 'FLAMES.Pipeline'
resume_FLAMES(pipeline)
```

Arguments

pipeline	A FLAMES.Pipeline object.

Value

An updated FLAMES.Pipeline object.

See Also

run_FLAMES to run the entire pipeline.

Examples

```
pipeline <- example_pipeline("BulkPipeline")
pipeline <- run_step(pipeline, "genome_alignment")
pipeline <- resume_FLAMES(pipeline)
```

run_FLAMES *Execute a FLAMES pipeline*

Description

This function runs the FLAMES pipeline. It will run all steps in the pipeline.

Usage

```
run_FLAMES(pipeline, overwrite = FALSE)

## S4 method for signature 'FLAMES.Pipeline'
run_FLAMES(pipeline, overwrite = FALSE)
```

Arguments

pipeline	A FLAMES.Pipeline object.
overwrite	(optional) If TRUE, the pipeline will be re-run even if some steps are already completed.

Value

An updated FLAMES.Pipeline object.

See Also

resume_FLAMES to resume a pipeline from the last completed step.

Examples

```
pipeline <- example_pipeline("BulkPipeline")
pipeline <- run_FLAMES(pipeline)
```

run_step *Execute a single step of the FLAMES pipeline*

Description

This function runs the specified step of the FLAMES pipeline.

Usage

```
run_step(pipeline, step, disable_controller = TRUE)

## S4 method for signature 'FLAMES.Pipeline'
run_step(pipeline, step, disable_controller = TRUE)
```

Arguments

pipeline	A FLAMES.Pipeline object.
step	The step to run. One of "barcode_demultiplex", "genome_alignment", "gene_quantification", "isoform_identification", "read_realignment", or "transcript_quantification".
disable_controller	(optional) If TRUE, the step will be executed in the current R session, instead of using crew controllers.

Value

An updated FLAMES.Pipeline object.

See Also

run_FLAMES to run the entire pipeline. resume_FLAMES to resume a pipeline from the last completed step.

Examples

```
pipeline <- example_pipeline("BulkPipeline")
pipeline <- run_step(pipeline, "genome_alignment")
```

---

scmixology_lib10 *scMixology short-read gene counts - sample 2*

Description

Short-read gene counts from long and short-read single cell RNA-seq profiling of human lung adenocarcinoma cell lines using 10X version 2 chemstry. See Tian, L. et al. Comprehensive characterization of single-cell full-length isoforms in human and mouse with long-read sequencing. Genome Biology 22, 310 (2021).

Usage

```
scmixology_lib10
```

Format

'scmixology_lib10' A SingleCellExperiment with 7,240 rows and 60 columns:

Value

A SingleCellExperiment object

Source

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE154869>

scmixology_lib10_transcripts *scMixology long-read transcript counts - sample 2*

Description

long-read transcript counts from long and short-read single cell RNA-seq profiling of human lung adenocarcinoma cell lines using 10X version 2 chemstry. See Tian, L. et al. Comprehensive characterization of single-cell full-length isoforms in human and mouse with long-read sequencing. Genome Biology 22, 310 (2021).

Usage

```
scmixology_lib10_transcripts
```

Format

'scmixology_lib10_transcripts' A `SingleCellExperiment` with 7,240 rows and 60 columns:

Value

A `SingleCellExperiment` object

Source

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE154869>

scmixology_lib90 *scMixology short-read gene counts - sample 1*

Description

Short-read single cell RNA-seq profiling of human lung adenocarcinoma cell lines using 10X version 2 chemstry. Single cells from five human lung adenocarcinoma cell lines (H2228, H1975, A549, H838 and HCC827) were mixed in equal proportions and processed using the Chromium 10X platform, then sequenced using Illumina HiSeq 2500. See Tian L, Dong X, Freytag S, Lê Cao KA et al. Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments. Nat Methods 2019 Jun;16(6):479-487. PMID: 31133762

Usage

```
scmixology_lib90
```

Format

'scmixology_lib90' A `SingleCellExperiment`

Value

A SingleCellExperiment object

Source

<https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE126906>

---

sc_DTU_analysis *FLAMES Differential Transcript Usage Analysis*

Description

Differential transcription usage testing for single cell data, using colLabels as cluster labels.

Usage

```
sc_DTU_analysis(
  sce,
  gene_col = "gene_id",
  min_count = 15,
  threads = 1,
  method = "trascript usage permutation",
  permuations = 1000
)
```

Arguments

sce	The SingleCellExperiment object, with transcript counts in the counts slot and cluster labels in the colLabels slot.
gene_col	The column name in the rowData slot of sce that contains the gene ID / name. Default is "gene_id".
min_count	The minimum total counts for a transcript to be tested.
threads	Number of threads to use for parallel processing.
method	The method to use for testing, listed in details.
permuations	Number of permutations for permutation methods.

Details

Genes with more than 2 isoforms expressing more than min_count counts are selected for testing with one of the following methods:

trascript usage permutation Transcript usage are taken as the test statistic, cluster labels are permuted to generate a null distribution.

chisq Chi-square test of the transcript count matrix for each gene.

Adjusted P-values were calculated by Benjamini–Hochberg correction.

Value

a `tibble` containing the following columns:

p.value - the raw p-value

adj.p.value - multiple testing adjusted p-value

cluster - the cluster where DTU was observed

transcript - rowname of `sce`, the DTU isoform

transcript_usage - the transcript usage of the isoform in the cluster

Additional columns from `method = "trascript usage permutation"`:

transcript_usage_elsewhere - transcript usage in other clusters

usage_difference - the difference between the two transcript usage

permuted_var - the variance of usage difference in the permuted data

Additional columns from `method = "chisq"`:

X_value - the test statistic

df - the degrees of freedom

expected_usage - the expected usage (mean across all clusters)

usage_difference - the difference between the observed and expected usage

The table is sorted by P-values.

Examples

```
outdir <- tempfile()
dir.create(outdir)
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)

sce <- FLAMES::sc_long_pipeline(
  genome_fa = genome_fa,
  fastq = system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  outdir = outdir,
  barcodes_file = bc_allow,
  config_file = create_config(
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex"
  )
```

```
)
group_anno <- data.frame(barcode_seq = colnames(sce), groups = SingleCellExperiment::counts(sce)["ENSMUST0000016
SingleCellExperiment::colLabels(sce) <- group_anno$groups
# DTU with permutation testing:
sc_DTU_analysis(sce, min_count = 1, method = "trascript usage permutation")
# now try with chisq:
sc_DTU_analysis(sce, min_count = 1, method = "chisq")
```

`sc_gene_entropy` *Compute Gene Isoform Entropy Matrix*

Description

Calculates normalized Shannon entropy for gene isoform expression across cells. Higher entropy indicates more diverse isoform usage, lower entropy indicates dominance by fewer isoforms.

Usage

```
sc_gene_entropy(
  sce,
  assay = "counts",
  gene_col = "gene_id",
  alpha = .Machine$double.xmin,
  min_counts_per_cell = 5,
  isoform_min_pct_cells = 0.05,
  isoform_cumulative_pct = 0.95,
  min_cell_fraction = 0.25,
  threads = 1,
  show_progress = interactive()
)
```

Arguments

`sce`	A `SingleCellExperiment` object
`assay`	Name of assay containing isoform counts (default: "counts")
`gene_col`	Column name in rowData containing gene identifiers (default: "gene_id")
`alpha`	Pseudocount added to avoid log(0) (default: .Machine$double.xmin)
`min_counts_per_cell`	Minimum total gene counts per cell to include (default: 5)
`isoform_min_pct_cells`	Minimum fraction of cells expressing each isoform (default: 0.05)
`isoform_cumulative_pct`	Keep top isoforms contributing to this cumulative proportion (default: 0.95)
`min_cell_fraction`	Minimum fraction of cells with valid entropy per gene (default: 0.25)
`threads`	Number of threads for parallel processing (default: 1)
`show_progress`	Logical indicating whether to show progress (default: TRUE if interactive)

Value

Matrix with genes as rows and cells as columns containing normalized entropy values (0-1).

Examples

```
sce <- scuttle::mockSCE(ncells = 50, ngenes = 30)
SummarizedExperiment::rowData(sce)$gene_id <- sort(
  paste0("gene", sample(1:9, nrow(sce), replace = TRUE))
)
res <- sc_gene_entropy(sce, threads = 2)
```

---

sc_genotype *Genotype a single-cell mutation*

Description

A simplistic function to genotype a single-cell mutation at a given position. It filters the SNPs table for the given reference and alternative alleles, and determines the genotype based on the allele counts and percentages.

Usage

```
sc_genotype(
  snps_tb,
  ref,
  alt,
  seqname,
  pos,
  alt_min_count = 2,
  alt_min_pct = 0.1,
  ref_min_count = 1,
  ref_min_pct = 1
)
```

Arguments

snps_tb	tibble: the SNPs table, output from sc_mutations.
ref	character(1): the reference allele.
alt	character(1): the alternative allele.
seqname	character(1): the chromosome name of the position.
pos	integer(1): the position of the mutation, 1-based.
alt_min_count	integer(1): minimum UMI count of the alternative allele to call it "alt".
alt_min_pct	numeric(1): minimum percentage of the alternative allele to call it "alt".
ref_min_count	integer(1): minimum UMI count of the reference allele to call it "ref".
ref_min_pct	numeric(1): minimum percentage of the reference allele to call it "ref".

Value

A tibble with columns: barcode, allele_count_ref, pct_ref, allele_count_alt, pct_alt, genotype.

Examples

```
# get the SNPs table from sc_mutations
example(sc_mutations)
genotype_tb <- snps_tb |>
  sc_genotype(
    ref = "G", alt = "A", seqname = "chr14", pos = 1260,
    alt_min_count = 2, alt_min_pct = 0.1,
    ref_min_count = 1, ref_min_pct = 1
  )
dplyr::count(genotype_tb, genotype)
head(genotype_tb)
```

---

sc_impute_transcript *Impute missing transcript counts*

Description

Impute missing transcript counts using a shared nearest neighbor graph

Usage

```
sc_impute_transcript(combined_sce, dimred = "PCA", ...)
```

Arguments

combined_sce	A SingleCellExperiment object with gene counts and a "transcript" altExp slot.
dimred	The name of the reduced dimension to use for building the shared nearest neighbor graph.
...	Additional arguments to pass to scran::buildSNNGraph. E.g. k = 30.

Details

For cells with NA values in the "transcript" altExp slot, this function imputes the missing values from cells with non-missing values. A shared nearest neighbor graph is built using reduced dimensions from the SingleCellExperiment object, and the imputation is done where the imputed value for a cell is the weighted sum of the transcript counts of its neighbors. Imputed values are stored in the "logcounts" assay of the "transcript" altExp slot. The "counts" assay is used to obtain logcounts but left unchanged.

Value

A SingleCellExperiment object with imputed logcounts assay in the "transcript" altExp slot.

Examples

```
sce <- SingleCellExperiment::SingleCellExperiment(assays = list(counts = matrix(rpois(50, 5), ncol = 10)))
long_read <- SingleCellExperiment::SingleCellExperiment(assays = list(counts = matrix(rpois(40, 5), ncol = 10)))
SingleCellExperiment::altExp(sce, "transcript") <- long_read
SingleCellExperiment::counts(SingleCellExperiment::altExp(sce))[,1:2] <- NA
SingleCellExperiment::counts(SingleCellExperiment::altExp(sce))
imputed_sce <- sc_impute_transcript(sce, k = 4)
SingleCellExperiment::logcounts(SingleCellExperiment::altExp(imputed_sce))
```

---

sc_long_multisample_pipeline

Pipeline for Multi-sample Single Cell Data (deprecated)

Description

This function is deprecated. Please use `MultiSampleSCPipeline`.

Usage

```
sc_long_multisample_pipeline(
  annotation,
  fastqs,
  outdir,
  genome_fa,
  minimap2 = NULL,
  barcodes_file = NULL,
  expect_cell_numbers = NULL,
  config_file = NULL
)
```

Arguments

`annotation`	The file path to the annotation file in GFF3 format
`fastqs`	The file path to input fastq file
`outdir`	The path to directory to store all output files.
`genome_fa`	The file path to genome fasta file.
`minimap2`	Path to minimap2, optional.
`barcodes_file`	The file with expected cell barcodes, with each barcode on a new line.
`expect_cell_numbers`	The expected number of cells in the sample. This is used if `barcodes_file` is not provided. See `BLAZE` for more details.
`config_file`	File path to the JSON configuration file.

Value

A list of `SingleCellExperiment` objects, one for each sample.

See Also

`MultiSampleSCPipeline` for the new pipeline interface, `SingleCellPipeline` for single-sample pipeline, `BulkPipeline` for bulk long data.

Examples

```
reads <- ShortRead::readFastq(
  system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES")
)
outdir <- tempfile()
dir.create(outdir)
dir.create(file.path(outdir, "fastq"))
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample1.fq.gz"), mode = "w", full = FALSE)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads[1:100],
  file.path(outdir, "fastq/sample2.fq.gz"), mode = "w", full = FALSE)
reads <- reads[-(1:100)]
ShortRead::writeFastq(reads,
  file.path(outdir, "fastq/sample3.fq.gz"), mode = "w", full = FALSE)

sce_list <- FLAMES::sc_long_multisample_pipeline(
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  fastqs = c("sampleA" = file.path(outdir, "fastq"),
    "sample1" = file.path(outdir, "fastq", "sample1.fq.gz"),
    "sample2" = file.path(outdir, "fastq", "sample2.fq.gz"),
    "sample3" = file.path(outdir, "fastq", "sample3.fq.gz")),
  outdir = outdir,
  genome_fa = genome_fa,
  barcodes_file = rep(bc_allow, 4),
  config_file = create_config(
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex"
  )
)
```

`sc_long_pipeline` *Pipeline for Single Cell Data (deprecated)*

Description

This function is deprecated. Please use [SingleCellPipeline()] instead.

Usage

```
sc_long_pipeline(
  annotation,
  fastq,
  outdir,
  genome_fa,
  minimap2 = NULL,
  barcodes_file = NULL,
  expect_cell_number = NULL,
  config_file = NULL
)
```

Arguments

`annotation`	The file path to the annotation file in GFF3 format
`fastq`	The file path to input fastq file
`outdir`	The path to directory to store all output files.
`genome_fa`	The file path to genome fasta file.
`minimap2`	Path to minimap2, optional.
`barcodes_file`	The file with expected cell barcodes, with each barcode on a new line.
`expect_cell_number`	The expected number of cells in the sample. This is used if `barcodes_file` is not provided. See `BLAZE` for more details.
`config_file`	File path to the JSON configuration file.

Value

A `SingleCellPipeline` object containing the transcript counts.

See Also

`SingleCellPipeline` for the new pipeline interface, `BulkPipeline` for bulk long data, `MultiSampleSCPipeline` for multi sample single cell pipelines.

Examples

```
outdir <- tempfile()
dir.create(outdir)
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
sce <- FLAMES::sc_long_pipeline(
  genome_fa = genome_fa,
  fastq = system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  outdir = outdir,
  barcodes_file = bc_allow,
  config_file = FLAMES::create_config(
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex"
  )
)
```

---

sc_mutations *Variant count for single-cell data*

Description

Count the number of reads supporting each variants at the given positions for each cell.

Usage

```
sc_mutations(bam_path, seqnames, positions, indel = FALSE, threads = 1)
```

Arguments

bam_path	character(1) or character(n): path to the bam file(s) aligned to the reference genome (NOT the transcriptome! Unless the postions are also from the transcriptome).
seqnames	character(n): chromosome names of the postions to count alleles.
positions	integer(n): positions, 1-based, same length as seqnames. The positions to count alleles.
indel	logical(1): whether to count indels (TRUE) or SNPs (FALSE).
threads	integer(1): number of threads to use. Maximum number of threads is the number of bam files * number of positions.

Value

A tibble with columns: allele, barcode, allele_count, cell_total_reads, pct, pos, seqname.

Examples

```
ppl <- example_pipeline("SingleCellPipeline")
ppl <- run_step(ppl, "barcode_demultiplex")
ppl <- run_step(ppl, "genome_alignment")
snps_tb <- sc_mutations(
  bam_path = ppl@genome_bam,
  seqnames = c("chr14", "chr14"),
  positions = c(1260, 2714), # positions of interest
  indel = FALSE
)
head(snps_tb)
snps_tb |>
  dplyr::filter(pos == 1260) |>
  dplyr::group_by(allele) |>
  dplyr::summarise(count = sum(allele_count)) # should be identical to samtools pileup
```

---

sc_plot_genotype *Plot genotype of single-cell data*

Description

Plot the genotype of single-cell data on a reduced dimension plot (e.g. UMAP).

Usage

```
sc_plot_genotype(
  sce,
  genotype_tb,
  reduced_dim = "UMAP",
  na_cell_col = "grey",
  na_cell_size = 0.1,
  na_cell_alpha = 0.1,
  ...
)
```

Arguments

sce	SingleCellExperiment: the single-cell experiment object with reduced dimensions.
genotype_tb	tibble: the genotype table, output from sc_genotype.
reduced_dim	character(1): the name of the reduced dimension to use for plotting.
na_cell_col	character(1): the color of the cells with no genotype.
na_cell_size	numeric(1): the size of the cells with no genotype.

na_cell_alpha	numeric(1): the alpha of the cells with no genotype.
...	additional arguments passed to geom_point for cells with genotype.

Value

A ggplot2 object with the genotype plotted on the reduced dimension.

Examples

```
ppl <- example_pipeline("SingleCellPipeline") |>
  run_FLAMES()
sce <- experiment(ppl) |>
 scuttle::logNormCounts() |>
 scater::runPCA() |>
 scater::runUMAP()
snps_tb <- sc_mutations(
  bam_path = ppl@genome_bam,
  seqnames = "chr14",
  positions = 2714
)
genotype_tb <- sc_genotype(
  snps_tb, ref = "C", alt = "T", seqname = "chr14", pos = 2714,
  alt_min_count = 2, alt_min_pct = 0.5, ref_min_count = 1, ref_min_pct = 1
)
sc_plot_genotype(
  sce, genotype_tb, na_cell_col = "black",
  na_cell_size = 0.5, na_cell_alpha = 0.7,
  size = 2
)
```

---

set_nested_param *Set Nested Configuration Parameter*

Description

Helper function to set a nested parameter in a configuration list using dot notation (e.g., "barcode_parameters.pattern.primer")

Usage

```
set_nested_param(config, param_path, value)
```

Arguments

config	Configuration list
param_path	Parameter path using dot notation
value	Value to set

Value

Modified configuration list

`show,FLAMES.Pipeline-method` *Show method for FLAMES.Pipeline*

Description

Displays the pipeline in a pretty format

Usage

```
## S4 method for signature 'FLAMES.Pipeline'
show(object)

## S4 method for signature 'FLAMES.SingleCellPipeline'
show(object)

## S4 method for signature 'FLAMES.MultiSampleSCPipeline'
show(object)
```

Arguments

`object`	An object of class ‘FLAMES.Pipeline‘

Value

None. Displays output to the console.

Examples

```
ppl <- example_pipeline()
show(ppl)
```

`SingleCellPipeline` *Pipeline for Single Cell Data*

Description

Semi-supervised isofrom detection and annotation for long read data. This variant is meant for single sample scRNA-seq data. Specific parameters can be configured in the config file (see `create_config`), input files are specified via arguments.

Usage

```
SingleCellPipeline(
  config_file,
  outdir,
  fastq,
  annotation,
  genome_fa,
  genome_mmi,
  minimap2,
  samtools,
  barcodes_file,
  expect_cell_number,
  controllers
)
```

Arguments

config_file	Path to the JSON configuration file. See create_config for creating one.
outdir	Path to the output directory. If it does not exist, it will be created.
fastq	Path to the FASTQ file or a directory containing FASTQ files. Each file will be processed as an individual sample.
annotation	The file path to the annotation file in GFF3 / GTF format.
genome_fa	The file path to the reference genome in FASTA format.
genome_mmi	(optional) The file path to minimap2's index reference genome.
minimap2	(optional) The path to the minimap2 binary. If not provided, FLAMES will use a copy from bioconda via basilisk.
samtools	(optional) The path to the samtools binary. If not provided, FLAMES will use a copy from bioconda via basilisk.
barcodes_file	The file with expected cell barcodes, with each barcode on a new line.
expect_cell_number	The expected number of cells in the sample. This is used if barcodes_file is not provided. See BLAZE for more details.
controllers	(optional, **experimental**) A crew_class_controller object for running certain steps

Details

By default the pipeline starts with demultiplexing the input fastq data. If the cell barcodes are known apriori (e.g. via coupled short-read sequencing), the barcodes_file argument can be used to specify a file containing the cell barcodes, and a modified Rcpp version of flexiplex will be used; otherwise, expect_cell_number need to be provided, and BLAZE will be used to generate the cell barcodes. The pipeline then aligns the reads to the genome using minimap2. The alignment is then used for isoform detection (either using FLAMES or bambu, can be configured). The reads are then realigned to the detected isoforms. Finally, a transcript count matrix is generated (either using FLAMES's simplistic counting or oarfish's Expectation Maximization algorithm, can be configured). The results can be accssed with experiment(pipeline). If the pipeline errored out / new steps were configured, it can be resumed by calling resume_FLAMES(pipeline)

Value

A `FLAMES.SingleCellPipeline` object. The pipeline can be run using `run_FLAMES(pipeline)`. The results can be accessed with `experiment(pipeline)`. The pipeline also outputs a number of output files into the given `outdir` directory. Some of these output files include:

matched_reads.fastq - fastq file with reads demultiplexed

align2genome.bam - sorted BAM file with reads aligned to genome

matched_reads_dedup.fastq - demultiplexed and UMI-deduplicated fastq file

transcript_assembly.fa - transcript sequence from the isoforms

isoform_annotated.filtered.gff3 - isoforms in gff3 format (also contained in the SingleCellExperiment)

realign2transcript.bam - sorted realigned BAM file using the transcript_assembly.fa as reference

See Also

`create_config` for creating a configuration file, `BulkPipeline` for bulk long data, `MultiSampleSCPipeline` for multi sample single cell pipelines.

Examples

```
outdir <- tempfile()
dir.create(outdir)
bc_allow <- file.path(outdir, "bc_allow.tsv")
genome_fa <- file.path(outdir, "rps24.fa")
R.utils::gunzip(
  filename = system.file("extdata", "bc_allow.tsv.gz", package = "FLAMES"),
  destname = bc_allow, remove = FALSE
)
R.utils::gunzip(
  filename = system.file("extdata", "rps24.fa.gz", package = "FLAMES"),
  destname = genome_fa, remove = FALSE
)
ppl <- SingleCellPipeline(
  config_file = create_config(
    outdir,
    pipeline_parameters.demultiplexer = "flexiplex",
    pipeline_parameters.do_gene_quantification = FALSE
  ),
  outdir = outdir,
  fastq = system.file("extdata", "fastq", "musc_rps24.fastq.gz", package = "FLAMES"),
  annotation = system.file("extdata", "rps24.gtf.gz", package = "FLAMES"),
  genome_fa = genome_fa,
  barcodes_file = bc_allow
)
ppl <- run_FLAMES(ppl)
experiment(ppl)
```

steps *Steps to perform in the pipeline*

Description

Steps to perform in the pipeline

Usage

```
steps(pipeline)

## S4 method for signature 'FLAMES.Pipeline'
steps(pipeline)
```

Arguments

pipeline	An object of class 'FLAMES.Pipeline'

Value

A named logical vector containing all possible steps for the pipeline. The names of the vector are the step names, and the values are logical indicating whether the step is configured to be performed.

Examples

```
ppl <- example_pipeline()
steps(ppl)
```

steps<- *Set steps to perform in the pipeline*

Description

Set steps to perform in the pipeline

Usage

```
steps(pipeline) <- value

## S4 replacement method for signature 'FLAMES.Pipeline'
steps(pipeline) <- value
```

Arguments

pipeline	An object of class 'FLAMES.Pipeline'
value	A named logical vector containing all possible steps for the pipeline. The names of the vector are the step names, and the values are logical indicating whether the step is configured to be performed.

Value

An pipeline of class 'FLAMES.Pipeline' with the updated steps.

Examples

```
ppl <- example_pipeline()
steps(ppl) <- c(
  barcode_demultiplex = TRUE,
  genome_alignment = TRUE,
  gene_quantification = TRUE,
  isoform_identification = FALSE,
  read_realignment = FALSE,
  transcript_quantification = TRUE
)
ppl
# or partially change a step:
steps(ppl)["read_realignment"] <- TRUE
ppl
```

---

weight_transcripts *Weight transcripts by read counts*

Description

Given a vector of read counts, return a vector of weights. The weights could be either the read counts themselves (`type = 'counts'`), a binary vector of 0s and 1s where 1s are assigned to transcripts with read counts above a threshold (`type = 'equal'`, `min_counts = 1000`), or a sigmoid function of the read counts (`type = 'sigmoid'`). The sigmoid function is defined as `1 / (1 + exp(-steepness/inflection * (x - inflection)))`.

Usage

```
weight_transcripts(
  counts,
  type = "sigmoid",
  min_counts = 1000,
  inflection_idx = 10,
  inflection_max = 1000,
  steepness = 5
)
```

Arguments

counts	numeric vector of read counts
type	string, one of 'counts', 'sigmoid', or 'equal'
min_counts	numeric, the threshold for the 'equal' type

inflection_idx	numeric, the index of the read counts to determine the inflection point for the sigmoid function. The default is 10, i.e. the 10th highest read count will be the inflection point.
inflection_max	numeric, the maximum value for the inflection point. If the inflection point according to the inflection_idx is higher than this value, the inflection point will be set to this value instead.
steepness	numeric, the steepness of the sigmoid function

Value

numeric vector of weights

Examples

```
weight_transcripts(1:2000)
par(mfrow = c(2, 2))
plot(
  1:2000, weight_transcripts(1:2000, type = 'sigmoid'),
  type = 'l', xlab = 'Read counts', ylab = 'Sigmoid weight'
)
plot(
  1:2000, weight_transcripts(1:2000, type = 'counts'),
  type = 'l', xlab = 'Read counts', ylab = 'Weight by counts'
)
plot(
  1:2000, weight_transcripts(1:2000, type = 'equal'),
  type = 'l', xlab = 'Read counts', ylab = 'Equal weights'
)
```

Index